Глава 8

Сбалансированные деревья

> Всякое движение — это видимое нами стремление к недостающему равновесию. Все живое движется в поисках его, в поисках утраченной гармонии, в стремлении к совершенству, когда покой не есть отсутствие движения, но равнодействующая всех движений.
>
> *Делия Стейнберг Гусман*

8.1. АВЛ-деревья

> На свете нет кружева тоньше,— негромко сказал отец. И показал рукой вверх, где листва деревьев вплеталась в небо — или, может быть, небо вплеталось в листву?
>
> *Рей Брэдбери*

Основные понятия

Двоичное дерево считается *идеально сбалансированным*, если для каждой его вершины количества вершин в левом и правом поддеревьях различаются не более чем на 1.

Для одних и тех же данных, например для целых чисел от 1 до n (в зависимости от порядка их поступления на обработку), структура двоичного дерева поиска будет различной. Возможны варианты как идеально сбалансированного дерева, так и простого линейного списка, когда или все левые, или все правые ссылки вершин равны `nil`. В этом случае время вставки и удаления имеет оценку $O(n)$, а в случае идеальной сбалансированности — $O(\log_2 n)$.

Реализация операции восстановления идеальной сбалансированности при случайной вставке или удалении элемента из двоичного дерева поиска, разумеется, возможна, однако она достаточно сложна. Поэтому в информатике в свое время были предприняты попытки (и они продолжаются по сей день) создания и использования структур, для которых операции по восстановлению балансировки двоичного дерева были бы простыми и выполнялись достаточно быстро.

Изменим немного определение сбалансированности и дадим ему следующую формулировку: *дерево является сбалансированным тогда и только тогда, когда для каждой вершины высота ее двух поддеревьев различается не более чем на единицу*. Деревья, удовлетворяющие этому условию, называют *АВЛ-деревьями*. Эта структура данных была предложена Г. М. Адельсоном-Вельским и Е. М. Ландисом в 1962 г. (отсюда и название — «АВЛ-дерево»), и эти их исследования, вероятно, были первыми, посвященными данной проблематике.

Предположим, что у каждой вершины двоичного дерева поиска есть поле `bal` (баланс вершины), значение которого определяет разность высот ее поддеревьев (из высоты правого поддерева вычитается высота левого поддерева). Очевидно, что значение `bal` для сбалансированного дерева лежит в диапазоне от −1 до 1. На рис. 8.1 приведены примеры деревьев, где дерево, показанное на рис. 8.1*а*, не является АВЛ-деревом, а на рис. 8.1*б* — является им. Заметим, что оба этих дерева не являются идеально сбалансированными.

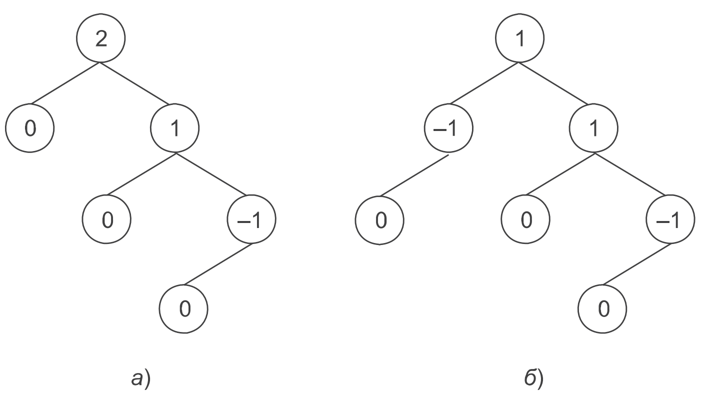

Рис. 8.1. Деревья: *а*) не АВЛ-дерево; *б*) АВЛ-дерево. В вершинах указаны разности высот поддеревьев

Работа с АВЛ-деревом (основная часть логики по вставке и удалению элементов) практически совпадает с операциями для двоичного дерева поиска, за исключением восстановления балансировки дерева, поэтому подробно мы рассмотрим здесь именно последнюю. Но прежде для большей конкретности изложения давайте введем описание дерева и опишем вспомогательные процедуры.

Описание вершины дерева, по аналогии с двоичным деревом поиска (см. раздел 5.2), имеет вид:

```
Type pt=^node;
         node = Record
           height:Word;
           data:Integer;
           left,right:pt ;
         End;
Var first:pt;
```

Как видим, здесь в описание вершины введено новое поле — высота дерева (height). Его назначение мы раскроем позднее, а сейчас лишь отметим, что именно введение этого поля (а не поля bal) позволяет упростить логику как изложения материала, так и программного кода.

Вычисление высоты дерева реализуется процедурой MakeNewHeight, которой при вызове необходимо передать значение указателя на вершину, являющуюся корнем дерева (поддерева). В этой процедуре берется значение высоты левого поддерева вершины, затем значение высоты правого поддерева, определяется максимальное из этих двух чисел, а затем к нему прибавляется единица.

```
Procedure MakeNewHeight(q:pt);
{Вычисление высоты дерева (поддерева)}
  Var a, b:Word;
  Begin
    If (q^.left<>nil) Then a:=q^.left^.height
      Else a:=0;
    If (q^.right<>nil) Then b:=q^.right^.height
      Else b:=0;
    If a>b Then q^.height:=a+1
      Else q^.height:=b+1;
  End;
```

Тогда для вычисления баланса вершины достаточно найти разность значений высот ее правого и левого потомков:

```
Function GetBallance(q:pt):Integer;
{Вычисление разности высот левого и правого потомков
вершины}
  Var a,b:Word;
```

```
Begin
  If (q^.left<>nil) Then a:=q^.left^.height
    Else a:=0;
  If (q^.right<>nil) Then b:=q^.right^.height
    Else b:=0;
  GetBallance:=b-a;
End;
```

Рассмотрим теперь операции восстановления баланса. Пусть у нас есть фрагмент дерева, показанный на рис. 8.2*а*, после вставки в него элемента с ключом 60. Баланс вершины (ее адрес — значение указателя q) становится равным −2, и необходимо восстановить балансировку. Мы как бы берем дерево за его правый конец и тянем его вниз, причем гвоздик вбит под вершиной с ключом 100 — этот фрагмент дерева поворачивается, и такой поворот мы назовем *малым левым поворотом*. Сложность здесь заключается в том, что у вершины с ключом 80 появится три потомка (на рис. 8.2 это не показано), и необходимо правого потомка вершины с ключом 80 сделать левым потомком вершины с ключом 100. Естественно, что после корректировки адресов связи (значений указателей в вершинах деревьев) необходимо заново подсчитать значения height для вершин, участвующих в операции.

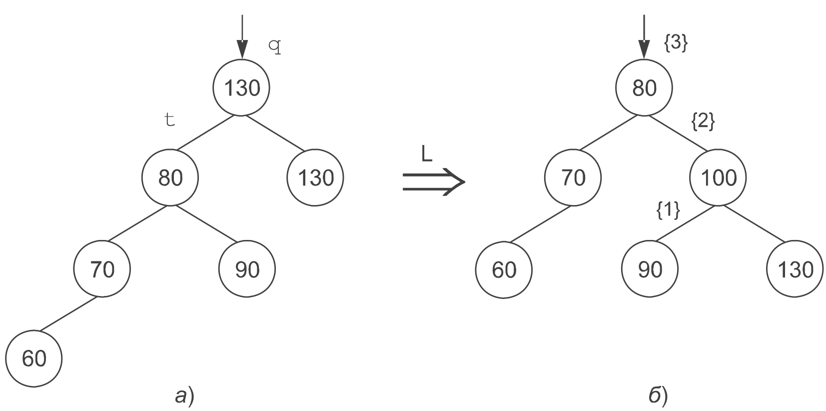

Рис. 8.2. Малый левый поворот

Процедура малого левого поворота на входе должна получить адрес вершины q, относительно которой нарушено условие баланса. Тогда мы запоминаем значение левой ссылки вершины (t), а на ее место записываем (первое действие) адрес правого поддерева вершины, переходящей в корень поддерева. На освобожденное место записываем (второе действие) значение q, пересчитываем высоты вершин с адресами q и t и в качестве выходного значения передаем вызывающей программе информацию, что новым корнем поддерева является вершина, которая ранее имела адрес t (третье действие).

```
Procedure TurnL(Var q:pt);
{Малый левый поворот: осуществляется для левого
поддерева вершины q}
  Var t:pt;
  Begin
    t:=q^.left;
    {Запоминаем значение левой ссылки}
    q^.left:=t^.right; {1}
    t^.right:=q;       {2}
    MakeNewHeight(q);
    MaketNewHeight(t);
    q:=t;              {3}
  End;
```

Аналогично осуществляется и поворот в другую сторону — *малый правый поворот* (рис. 8.3).

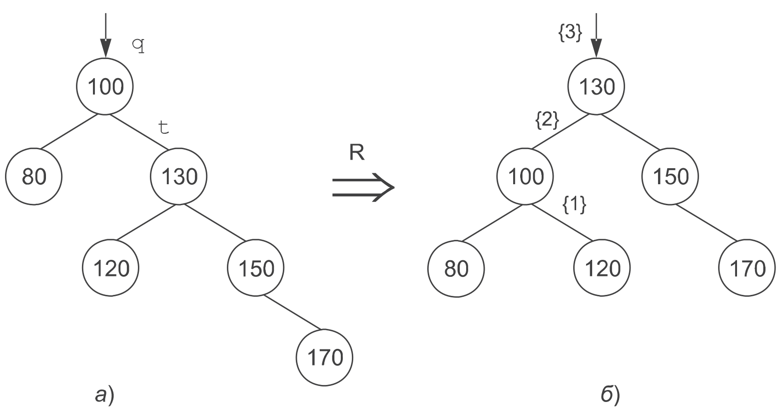

Рис. 8.3. Малый правый поворот

Процедура, реализующая это действие, повторяет процедуру TurnL с точностью до замены ссылок left на ссылки right и наоборот, что вполне естественно.

```
Procedure TurnR(Var q:pt);
{Малый правый поворот}
  Var t:pt;
  Begin
    t:=q^.right;
    q^.right:=t^.left; {1}
    t^.left:=q;        {2}
    MakeNewHeight(q);
    MakeNewHeight(t);
    q:=t;              {3}
  End;
```

Рассмотрим фрагмент дерева, показанный на рис. 8.4. Здесь нарушена балансировка в левом поддереве вершины с указателем q (например, если была выполнена вставка вершины с ключом 70; баланс вершины с ключом 100 равен −2). Выполняя малый левый поворот, мы не исправим эту ситуацию: в повернутом поддереве баланс вершины с ключом 60 равен 2. Вывод: левый поворот (опустим в его названии слово «малый», ибо «большого» поворота, как это сделано в работе Н. Вирта[1)], в нашем изложении не будет) относительно вершины с указателем q приводит к результату, только если у левого потомка вершины левое поддерево имеет бóльшую высоту, чем правое.

На рис. 8.5 приведена схема разрешения возникшего затруднения. Относительно левого потомка вершины с указа-

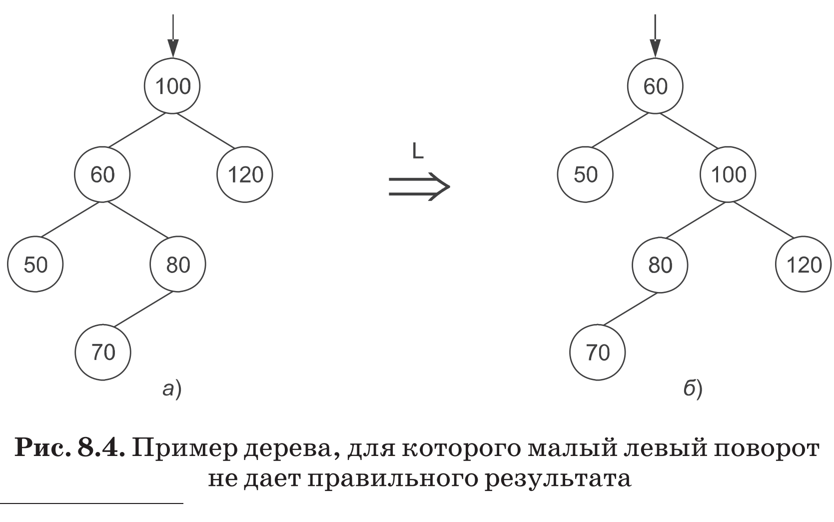

Рис. 8.4. Пример дерева, для которого малый левый поворот не дает правильного результата

---

1) *Вирт Н.* Алгоритмы+структуры данных=программы. М.: Мир, 1985. С. 248–260.

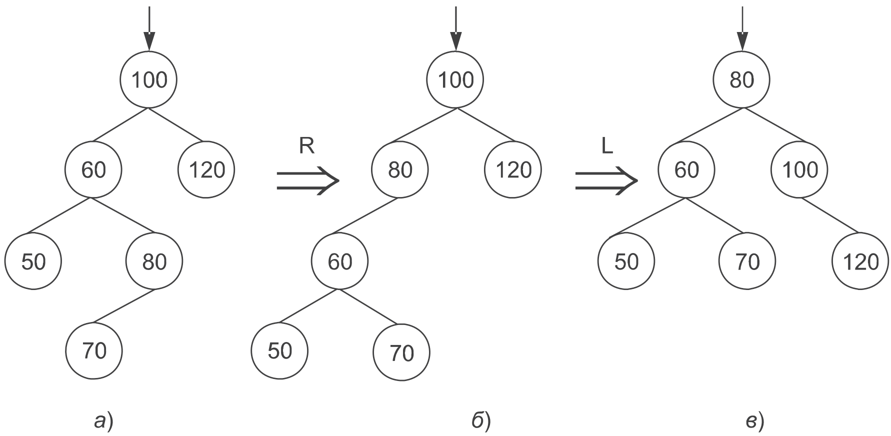

Рис. 8.5. Выполнение двух поворотов — правого и левого — восстанавливает балансировку дерева

телем `q` мы, несмотря на то что эта часть дерева уже сбалансирована, выполняем правый поворот (8.5*б*). Дерево останется несбалансированным, но теперь у левого потомка вершины `q` левое поддерево будет иметь бо́льшую высоту, и появится возможность выполнить левый поворот (8.5*в*), который приведет к нужному нам результату: балансировка дерева будет восстановлена.

Аналогично разрешается ситуация и когда правое поддерево вершины с указателем `q` увеличивается по высоте — значение `bal` равно 2 (или когда левое поддерево уменьшается по высоте на единицу при `bal` = 1).

Перечисленными выше вариантами исчерпываются все случаи нарушения балансировки, и теперь у нас есть возможность реализовать полный текст логики с необходимыми и достаточными комментариями. Отметим, что она одна и та же как при вставке элемента в АВЛ-дерево, так и при удалении из него элемента (конечно, если при этом нарушается балансировка).

```
Procedure Ballance(Var q:pt);
  Var bal, old_height:Integer;
  Begin
    old_height:=q^.height;
    {Сохранение высоты поддерева}
    MakeNewHeight(q);
    {Вычисление новой высоты с учетом изменений}
```

```
  bal:=GetBallance(q);
  {Вычисление разности высот}
  If (bal>1) Then Begin
  {Правое поддерево выше допустимого уровня}
    If (GetBallance(q^.right)<0) Then
      TurnL(q^.right);
      {Если у правого потомка q, как корня
      поддерева, левое  поддерево имеет
      большую высоту, то необходимо выполнить
      левый поворот относительно этой вершины,
      хотя эта часть дерева уже сбалансирована.
      После этого становится возможным правый
      поворот относительно q}
    TurnR(q);
    {Выполняется правый поворот}
    If (q^.height=old_height) Then h:=False;
    {Сбрасывание флага. h – глобальная переменная
    для фиксации факта вставки или удаления
    элемента, когда, возможно, требуется
    балансировка дерева. Если новая высота q
    совпала со старой, а последняя соответствовала
    сбалансированному дереву, то дальнейшая
    балансировка не требуется}
  End
  Else
    If (bal<-1)Then Begin
    {Левое поддерево выше допустимого}
      If (GetBallance(q^.left)>0) Then
        TurnR(q^.left);
        {Если у левого потомка q, как корня
        поддерева, правое  поддерево имеет
        большую высоту, то выполнение сразу левого
        поворота относительно q не приводит
        к нужному результату. Относительно этого
        потомка, как корня поддерева, несмотря
        на сбалансированность этой части дерева,
        необходимо выполнить правый поворот}
      TurnL(q); {Выполняется левый поворот}
      If (q^.height=old_height) Then h:=False;
    {Сброс флага}
  End;
End;
```

Хотя логика процедур вставки и удаления элементов АВЛ-дерева практически совпадает с описанной в разделе 5.2 для двоичного дерева поиска, приведем текст процедуры `Insert`, чтобы указать место и время вызова процедуры `Balance`. Восстановление балансировки дерева осуществляется на выходе из рекурсии, т. е. мы как бы идем снизу вверх по дереву, последовательно проверяя нарушение баланса вершин и восстанавливая балансировку, пока не достигнем корня дерева.

```
Procedure Insert(x:Integer; Var q:pt);
  Begin
    If (q=nil) Then Begin
    {Место для вставки вершины найдено}
      New(q);
      h:=True;
      {Признак: элемент вставлен и необходима
      проверка того, что дерево осталось
      сбалансированным}
      With q^ Do Begin
        data:=x;
        left:=nil;
        right:=nil;
        height:=1;
      End;
    End
    Else
      If x<q^.data Then Begin
      {Новая вершина должна принадлежать левому
      поддереву данной вершины}
        Insert(x, q^.left);
        If h Then Ballance(q);
        {h - глобальная переменная, ее инициализация
        значением False осуществляется в основной
        программе}
      End
      Else Begin
      {Или новая вершина должна принадлежать левому
      поддереву данной вершины}
        Insert(x, q^.right);
        If h Then Ballance(q);
      End;
  End;
```

Математическое отступление. Пусть n_h — минимальное кочичество узлов в АВЛ-дереве высоты h. Тогда $n_0 = 1$, $n_1 = 2$, $n_2 = 4$, $n_h = n_{h-1} + n_{h-2} + 1$ при $h \geqslant 2$.

Теорема. Для любого $h \geqslant 3$ выполняется неравенство $n_h \geqslant \alpha^{h+1}$, где $\alpha = (1 + \sqrt{5})/2$ — положительный корень уравнения $x^2 - x - 1 = 0$.

Доказательство этой теоремы выполняется по индукции. Проверяется базис n_3 и n_4. Предполагается, что неравенство верно для интервала значений до некоторого значения t включительно, и выводится его верность для значения $t + 1$.

Из теоремы следует, что для любого АВЛ-дерева высоты h с n узлами выполняется соотношение $h + 1 < \log_\alpha n = \log_\alpha 2 \cdot \log_2 n \approx 1.44 \cdot \log_2 n$. Таким образом, время выполнения операций вставки и удаления имеет порядок $O(\log_2 n)$.

Упражнения

1. Можно ли изменить последовательность вызовов процедуры `MakeNewHeight` в процедурах `TurnL` и `TurnR`? Обоснуйте свой ответ.

2. Приведите пример дерева, для которого требуется выполнить последовательность из левого и правого поворотов для восстановления балансировки. Объясните, почему один только правый поворот не восстанавливает баланс этого дерева.

3. Приведите пример дерева, в котором после удаления элемента необходимо выполнить левый (правый) поворот.

4. Приведите пример дерева, в котором после удаления элемента необходимо выполнить последовательность из левого и правого поворотов.

5. Приведите пример дерева, в котором после удаления элемента необходимо выполнить последовательность из правого и левого поворотов.

6. Напишите процедуру удаления элемента из АВЛ-дерева. Выполните ее трассировку для различных типов поворотов.

7. Выполните трассировку процедуры `PrintStep` для дерева на рис. 8.2*б*. Первый вызов — `PrintStep(first,1)`. Какая информация и в какой последовательности (в каком виде) будет выведена в результате работы процедуры?

```
Procedure PrintStep(q:pt; r:Integer);
  Begin
    If (q<>nil) Then Begin
      PrintStep(q^.right,r+8);
      WriteLn(q^.data : r,',',q^.height : 2, ' ');
      PrintStep(q^.left,r+8);
     End;
  End;
```

8. Напишите программу для работы с АВЛ-деревом. Вставка, удаление элементов должны осуществляться случайным образом. Обеспечьте наглядный вывод АВЛ-дерева, отражающий его структуру.

9. Найдите АВЛ-дерево с 12 вершинами, имеющее максимальную высоту среди всех АВЛ-деревьев с 12 вершинами. В какой последовательности необходимо вставлять вершины (с помощью процедуры `Insert`), чтобы было получено такое дерево?

10*. Найдите такую последовательность из n включаемых в АВЛ-дерево элементов, чтобы повороты: левый, правый, левый —правый и правый — левый выполнялись, по крайней мере, один раз. Какова минимальная длина n такой последовательности?

11*. Найдите АВЛ-дерево с ключами 1, 2, ..., n и такую перестановку этих ключей, чтобы при удалении элементов из этого дерева выполнялись повороты: левый, правый, левый — правый и правый — левый, по крайней мере, один раз. Какова будет последовательность с минимальной длиной n?